COMPOPT

- For Mainframes
- For UNIX/OpenVMS
- For Windows

COMPOPT for Mainframes

```
COMPOPT [option=value]
```

This system command is used to set various compilation options. The options are evaluated when a Natural programming object is compiled.

option=value

The keywords for the individual options are shown on the Compilation Options screen and are described in the section Options.

The settings assigned to a compiler option are in effect until you issue the next LOGON command to another library. At LOGON, the default settings set with the macro NTCMPO and/or profile parameterCMPO will be resumed.

This section covers the following topics:

- General Information on Compiler Options
- Options

General Information on Compiler Options

You can specify compiler parameters on different levels:

- The default settings of the individual parameters are set with the macro NTCMPO in the Natural parameter module NATPARM.
- At session start, you can override the compiler parameters with the profile parameter CMPO.
- During an active Natural session, there are two ways to change the compiler parameters with the COMPOPT command: either directly using command assignment (COMPOPT *option=value*) or by issuing the COMPOPT command without parameters which displays the Compilation Options screen. For further in formation, see the section Options. The settings assigned to a compiler option are in effect until you issue the next LOGON command to another library. At LOGON, the default settings set with the macroNTCMPO and/or profile parameter CMPO (see above) will be resumed.
- In a Natural programming object (for example: program, subprogram), you can set compiler parameters with the OPTIONS statement.
  Example:

  ```
  0010 OPTIONS KCHECK=ON
  0020 WRITE ’Hello World’
  0030 END
  ```

  The compiler options defined in an OPTIONS statement will only affect the compilation of this programming object, but do not update settings set with the command COMPOPT.

Options

If you issue the COMPOPT command without parameters, the Compilations Options screen appears.

- KCHECK - Keyword Checking
- PCHECK - Parameter Checking for CALLNAT Statements
- DBSHORT - Interpretation of Database Short Field Names
- PSIGNF - Internal Representation of Positive Sign of Packed Numbers
- TSENABL - Applicability of TS Profile Parameter
- GFID - Generation of Global Format IDs
- LOWSRCE - Allow Lower-Case Source
- FINDMUN - Detect Inconsistent Comparison Logic in FIND Statements
- V22COMP - Compatibility Option for Inconsistent Version 2.2 Syntax

KCHECK - Keyword Checking

ON	Programming objects will be checked for Natural statement keywords. Variable names which are reserved Natural keywords are rejected. The section Keywords and Reserved Words in the Natural Reference documentation contains a list of all Natural keywords and reserved words, in which the statement keywords affected by this option are marked.
OFF	No keyword check is performed. This is the default.

PCHECK - Parameter Checking for CALLNAT Statements

ON	It will be checked if the number of the parameters specified in a CALLNAT statement corresponds with the number of parameters in the subprogram to be invoked - provided that subprogram already exists (if the subprogram does not exist, this option has no effect).
OFF	No parameter check is performed. This is the default.

DBSHORT - Interpretation of Database Short Field Names

ON	Database field names in programming objects are considered long names (as defined in the corresponding DDM) - except 2-character field names, which are considered short names (as used by the underlying database system). This is the default. **Note:** Even if DBSHORT=ON, the use of a short database field is not allowed in the following cases: ● if a DEFINE DATA LOCAL is specified in a program, ● in Natural for non-mainframe platforms, ● when Natural Security is installed
OFF	All database field names in programming objects are considered long names, regardless of their length. This avoids possible misinterpretations of database field names in programs. You can use this setting to disallow the use of short names in general.

Background Information

When the Natural compiler resolves a database field (that is, a field defined in a DDM), the length of the field name is used to decide if the identifier represents a "db-short-name" or a "db-long-name". When the field name length is 2 characters a "db-short-name" reference is assumed, whereas all other identifiers are treated as "db-long-names".

According to the general Natural rules, you may not use "db-short-names" in your program, if a DEFINE DATA LOCAL was specified; neither to create the field list in the view definition nor in a search expressions of a FIND statement or to specify a read sequence control field in a READ or HISTOGRAM statement. All these restrictions are controlled by the Natural compiler, regardless if option DBSHORT is ON or OFF.

The purpose of DBSHORT is to change the compiler's behavior as follows:

if set to ON ,	everything works as described above.
if set to OFF,	every database field identifier is regarded as a "db-long-name", no matter of how many characters it consist. In other words, only the long name column in the DDM (captioned as "Name" in a DDM display) is considered to locate the referenced field and the DDM short names (captioned as "DB" in a DDM display) are completely disregarded.

The main reason for using DBSHORT=ON is when you have long names defined in a DDM with only 2 byte identifier length. At DDM generation, you may only create 2 byte long-names if the underlying database (you access with this DDM) is SQL (e.g. DB2). For all other database types, the attempt to define a long-field with 2 byte name length results in error SYSDDM4219 (SYSDDM utility).

However, when DBSHORT=OFF is set, the compiler does not check db-short-names in a DDM. This leads to syntax error NAT0981 if a db-short-field is used in a program.

PSIGNF - Internal Representation of Positive Sign of Packed Numbers

ON	The positive sign of a packed number is represented internally as H'F'. This is the default.
OFF	The positive sign of a packed number is represented internally as H'C'.

TSENABL - Applicability of TS Profile Parameter

This option determines whether the profile parameter TS (translate output for locations with non-standard lower-case usage) is to apply only to Natural system libraries (that is, libraries whose names begin with "SYS", except SYSTEM) or to all user libraries as well.

ON	The TS parameter applies to all libraries.
OFF	The TS parameter only applies to Natural system libraries. This is the default.

The setting of the TSENABL option is currently not honored when a Natural object is executed. This will be corrected in the next major relase of Natural.

GFID - Generation of Global Format IDs

This option allows you to control Natural's internal generation of global format IDs so as to influence Adabas's performance concerning the re-usability of format buffer translations.

ON	Global format IDs are generated for all views. This is the default.
VID	Global format IDs are generated only for views in local/global data areas, but not for views defined within programs.
OFF	No global format IDs are generated.

Rules for generating GLOBAL FORMAT-IDs in Natural Version 3.1.

Note: STOD is the return value of the store clock machine instruction (STCK).

For Natural nucleus internal system-file calls

`GFID=`*`abccddee`*

where	equals
a	x’F9’
b	x’22’ or x’21’ depending on DB statement
cc	physical database number (2 bytes)
dd	physical file number (2 bytes)
ee	number created by runtime (2 bytes)

For user programs or Natural utilities

a) GFID=*abbbbbbc* for file number equal to or less than 255 and Adabas Version lower than 6.2 (see NTDB macro).

where	equals
a	x’F8’ or x’F7’ or x’F6’
bbbbbb	byte 1-6 of STOD value
c	physical file number

b) GFID=*axbbbbbc* for file number greater than 255 and Adabas Version lower than 6.2.

where	equals
a	x’F8’ or x’F7’ or x’F6’
x	physical file number - high order byte
bbbbb	byte 2-6 of STOD value
c	physical file number - low order byte

c) GFID=*abbbbbbb* for Adabas Version 6.2 or higher.

where	equals

a	x’F8’ or x’F7’ or x’F6’
	where:
	F6=UPDATE SAME
	F7=HISTOGRAM
	F8=all others

bbbbbbb	byte 1-7 of STOD value

For details on global format IDs, see the Adabas documentation.

LOWSRCE - Allow Lower-Case Source

This option supports the use of lower or mixed-case program sources on mainframe platforms. It facilitates the transfer of programs written in mixed/lower-case characters from other platforms to a mainframe environment.

ON	Allows any kind of lower/upper-case characters in the program source.
OFF	Allows upper-case mode only. This requires keywords, variable names and identifier to be defined in upper case. This is the default.

When you use lower-case characters with LOWSRCE=ON consider the following:

- The syntax rules for variable names allow lower-case characters in subsequent positions. Therefore, you can define two variables, one written with lower-case characters and the other with upper-case characters. Example:

  ```
  DEFINE DATA LOCAL
  1 #Vari  (A20)
  1 #VARI  (A20)
  ```

  With LOWSRCE=OFF, these variables are treated as different variables.

  With LOWSRCE=ON, the compiler is **not** case sensitive and does not make a difference between lower/upper-case characters. This will lead to a syntax error because a duplicate definition of a variable is not allowed.
- Using the **session parameter** EM (Edit Mask) in an I/O statement or in a MOVE EDITED statement, there are characters which influence the layout of the data setting assigned to a variable (EM control characters), and characters which insert text fragments into the data setting. Example:

  ```
  #VARI :='1234567890'
    WRITE #VARI (EM=XXXXXxxXXXXX)
  ```

  With LOWSRCE=OFF, the output is 12345xx67890, because for alpha format variables only upper-case **X**, **H** and circumflex accent (ˆ) sign can be used.

  With LOWSRCE=ON, the output is 1234567890, because an **x** character is treated like an upper-case **X** and, therefore, interpreted as an EM control characters for that field format. To avoid this problem, enclose constant text fragments in apostrophes ('). Example:

  ```
  WRITE #VARI(EM=XXXXX'xx'XXXXX)
  ```

  The text fragment is **not** considered an EM control character, regardless of the LOWSRCE settings.

- Since all variable names are converted to upper-case characters with LOWSRCE=ON, the display of variable names in I/O statements (INPUT, WRITE or DISPLAY) differs. Example:

  ```
  MOVE 'ABC' to #Vari
    DISPLAY #Vari
  ```

  With LOWSRCE=OFF, the output is:

  ```
        #Vari
    --------------------

    ABC
  ```

With LOWSRCE=ON, the output is:

```
     #VARI
--------------------

ABC
```

FINDMUN - Detect Inconsistent Comparison Logic in FIND Statements

With Natural Version 2.3, the comparison logic for multiple-setting fields in the WITH clause of the FIND statement has been changed. This means that when Version 2.2 programs containing certain forms of FIND statements are compiled under Version 3.1, they will return different results. This option can be used to search for FIND statements whose WITH clauses use multiple-setting fields in a way that is no longer consistent with the enhanced Version 3.1 comparison logic.

ON	Error NAT0998 will be returned for every FIND statement of such form detected at compilation.
OFF	No search for such FIND statements will be performed. This is the default.

The comparison logic for multiple-value fields in the WITH clause of the FIND statement has been changed with Natural version 2.3 so as to be in line with the comparison logic in other statements (e.g. IF).

Four different forms of the FIND statement can be distinguished (the field MU in the following examples is assumed to be a multiple-value field):

1.

   ```
   FIND XYZ-VIEW WITH MU = 'A'
   ```

   With Version 2.2 and above, this statement returns records in which at least one occurrence of MU has the value "A".

2.

   ```
   FIND XYZ-VIEW WITH MU NOT EQUAL 'A'
   ```

   With Version 2.2, this statement returns records in which no occurrence of MU has the value "A" (same as 4.).
   With Version 2.3 and above, this statement returns records in which at least one occurrence of MU does not have the value "A".

3.

   ```
   FIND XYZ-VIEW WITH NOT MU NOT EQUAL 'A'
   ```

   With Version 2.2, this statement returns records in which **at least one occurrence** of MU has the value "A" (same as 1.).
   With Version 2.3 and above, this statement returns records in which **every occurrence** of MU has the value "A".

4.

   ```
   FIND XYZ-VIEW WITH NOT MU = 'A'
   ```

   With Version 2.2 and above, this statement returns records in which **no occurrence** of MU has the value "A".
   This means that if you newly compile under Version 2.3 existing Version 2.2 programs containing FIND statements of the forms **2.** and **3.**, they will return different results.

If you specify FINDMUN=ON, error NAT0998 will be returned for every FIND statement of form **2.** or **3.** detected at compilation.

Should you in these cases wish to continue to get the same results as with Version 2.2, you have to change the statements as follows:

In Form 2:

```
FIND XYZ-VIEW WITH MU NOT EQUAL 'A'
```

into

```
FIND XYZ-VIEW WITH NOT MU = 'A'
```

In Form 3:

```
FIND XYZ-VIEW WITH NOT MU NOT EQUAL 'A'
```

into

```
FIND XYZ-VIEW WITH MU = 'A'
```

V22COMP - Compatibility Option for Inconsistent Version 2.2 Syntax

⚠ This option will be available only for a limited period of time to allow a smooth transition. It will be removed again with a subsequent release of Natural.

The following inconsistent syntax constructions that were not intercepted by Version 2.2 lead to a syntax error with Version with Version 2.3 and above:

- DEFINE DATA: inconsistent number of decimal digits in constant setting,
- DEFINE DATA: redefinition of database array with variable index range,
- DEFINE WINDOW: specified window size smaller than minimum.

To allow you a smooth transition from Version 2.2, you can use this option.

ON	The above syntax constructions will **not** lead to a syntax error. Thus you are able to compile your existing programs with the current Natural version until you have adjusted them to the new syntax requirements.
OFF	The above syntax constructions will lead to a syntax error. This is the default.

Below, the above inconsistencies are explained in detail.

Decimal Digits of Constant Values

If the constant value specified after CONSTANT or INIT has more digits after the decimal point than the corresponding field, this does not lead to an error with Version 2.2. Now, such inconsistency leads to error NAT0094 at compilation.

Example:

```
DEFINE DATA LOCAL1 #FIELD (N2) INIT <12.25> /* no longer possibleEND-DEFINE
```

Redefinition of Database Arrays

To prevent referencing errors, it is no longer possible to specify a variable index range in the redefinition of a periodic-group field or multiple-value field.

Example:

```
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
2 SALARY (I:I+2)
2 REDEFINE SALARY
3 MYSALARY (P9/I:I+2) /* no longer possible
END-DEFINE
```

The above redefinition has to be changed to:

```
3 MYSALARY (P9/1:3)
```

DEFINE WINDOW

The minimum possible size of a window is 2 lines by 10 columns without frame, and 4 lines by 12 columns with frame. The size of a window can be specified in the SIZE clause of the DEFINE WINDOW statement.

With Version 2.2, a window size smaller than the minimum could be specified: at runtime, the size of the window was then automatically set to the minimum possible number of lines and/or columns. Now, the specification of too small a window size leads to error NAT1167 at compilation.

Example:

```
DEFINE WINDOW XYZ
SIZE 4 * 8 FRAMED OFF      /* Version 2.2: Size set to 4 * 10 at runtime.
                           /* Version 2.3 and above: Syntax error.
```

COMPOPT for UNIX/OpenVMS

```
COMPOPT [option = value]
```

This command is used to set various compilation options. The options are evaluated when a Natural programming object is compiled.

option=value

When you issue the COMPOPT command without parameters, a screen is displayed on which you can set the options described below. Alternatively, you can set the options directly with the COMPOPT command using the keywords shown both in brackets below and on the COMPTOPT screen.

Example:

```
COMPOPT DBSHORT=ON
```

Note:
The default settings of the individual options are set with the respective profile parameters in the Natural parameter module.

Interpretation of Database Short Field Names (DBSHORT):

ON	Database field names in programming objects are considered long names (as defined in the corresponding DDM) - except 2-character field names, which are considered short names (as used by the underlying database system).
OFF	All database field names in programming objects are considered long names, regardless of their length. This avoids possible misinterpretations of database field names in programs.

Generation of Global Format IDs (GFID):

This option allows you to control Natural's internal generation of global format IDs so as to influence Adabas's performance concerning the re-usability of format buffer translations.

ON	Global format IDs are generated for all views.
VID	Global format IDs are generated only for views in local/global data areas, but not for views defined within programs.
OFF	No global format IDs are generated.

For details on global format IDs, see the Adabas documentation.

Allow Old Version 2.2 Syntax (V22COMP):

This option will be available only for a limited period of time to allow a smooth transition to Version 3.1 syntax and above. It will be removed again with a subsequent release of Natural.

The following inconsistent syntax construction not intercepted by Version 2.2 lead to a syntax error with Version 3.1 and above:

- DEFINE DATA: inconsistent number of decimal digits in constant setting.

To allow you a smooth transition from Version 2.2 to Version 3.1, you can use this option.

ON	The above syntax constructions will **not** lead to a syntax error. Thus you are able to compile your existing programs under Version 3.1 until you have adjusted them to the Version 3.1 requirements.
OFF	The above syntax constructions will lead to a syntax error.

For details on the above inconsistencies, please refer to your Natural for UNIX/OpenVMS Version 4.1.2 Release Notes.

COMPOPT for Windows

```
COMPOPT [option = value]
```

This command is used to set various compilation options. The options are evaluated when a Natural programming object is compiled.

When you issue only the COMPOPT command itself, a screen is displayed where you can set the options described below.

Note:
The default settings of the individual options are set with the respective profile parameters in the Natural parameter module.
When you log on to another library, the COMPOPT options will be reset to their default settings.

Interpretation of Database Short Field Names - DBSHORT:

ON	Database field names in programming objects are considered long names (as defined in the corresponding DDM) - except 2-character field names, which are considered short names (as used by the underlying database system).
OFF	All database field names in programming objects are considered long names, regardless of their length. This avoids possible misinterpretations of database field names in programs.

Compatibility Option for Inconsistent Version 2.2 Syntax - V22COMP:

This option will be available for a limited period of time only to allow a smooth transition from Version 2.2. to Version 3.1 or above. It will be removed again with a subsequent release of Natural.

The following inconsistent syntax construction not intercepted by Version 2.2 leads to a syntax error with Version 3.1 or above.

- DEFINE DATA: inconsistent number of decimal digits in format definition and constant setting.

ON	The above syntax constructions will ***not*** lead to a syntax error. Thus you are able to compile your existing programs under version 3.1 or above until you have adjusted them to the version requirements.
OFF	The above syntax constructions will lead to a syntax error.

Generation of Global Format IDs - GFID:

This option enables you to control Natural's internal generation of global format IDs so as to influence Adabas's performance concerning the re-usability of format buffer translations.

ON	Global format IDs are generated for all views.
VID	Global format IDs are generated only for views in local/global data areas, but not for views defined within programs.
OFF	No global format IDs are generated.

For details on global format IDs, see the Adabas documentation.

option=value

Instead of changing an option on the screen, you can also specify it directly with the COMPOPT command. The keywords for the individual options are shown in parentheses (on the COMPOPT screen and in the above description).

Example:

COMPOPT DBSHORT=ON